On the Usefulness of Mapping Tables in APs

In her email of 9 Feb 1996 to the WG10 email exploder Yuhwei Yang (PDIT, USA), the former ISO/TC184/SC4/WG4 convener, stated very clearly what the intention was (and still is) which lead to the invention and usage of Mapping Tables in Application Protocols:

> The mapping table establishes the bridge to connect ARM to AIM. Ultimately, when we have sufficiently friendly tools available, an end user should not need to worry about the AIM structure at all. So, from this perspective the ARM is totally preserved.

Let me examine the scenario described here a little bit further. First of all, the availability of the “sufficiently friendly tools” is surely a necessary precondition for this scenario to be implementable and, hence, usable with affordable costs. But other preconditions are necessary as well:

- The ARM model has to be available as part of the AP document. Currently no computer readable form of the ARM is documented and publicly known.
- The ARM model has to be defined using a computer sensible language, which is powerful enough e. g. to describe necessary consistency conditions for the data. EXPRESS is such a language. Currently some ARMs are written in languages other than EXPRESS.
- The ARM model has to be completely defined. Currently some parts (like geometry) of the ARM are only giving an idea of the data needed, but the precise specification is not available on the ARM level.
- The mapping between ARM and AIM has to be precisely defined in terms of how the data **instances** map onto each other. Currently there are a number of cases where the relation on the entity instance level is not well defined, because description and considerations leading to it have been performed on the entity type level only, disregarding the entity instance level.
- The mapping between ARM and AIM has to be defined using a computer sensible language, which is powerful enough and which is suitably linked with EXPRESS. Such a language does not yet exist. Currently the mapping between ARM and AIM is defined using a pseudo-formalism for which not even a formal grammar exists, neither does well defined semantics.
- The mapping between ARM and AIM has to be defined such that a two way map is described or implied, i. e. such that the translation of AIM data into ARM data is defined as well, for all cases of a legal AIM population. The image of this inverse map may be an erroneous state of the ARM (not mappable or not allowed) for some legal AIM population. If this is possible, it has to be defined precisely for which AIM populations an erroneous state occurs.
- The implementation costs for the “sufficiently friendly tools” and the market size of them has to be such that there is a fair chance of making profit for the friendly tool vendor.

Given all these preconditions be satisfied (and potentially more which I forgot), in the end the scenario worked as follows: From the application programmer's perspective the AP is just the ARM, he or she doesn't care about the AIM at all, because the data are sent and received as ARM conforming data. Application programs are implemented on the basis of the ARM data model, and the rest of the AP is considered a black box, like the software and hardware details of the Internet is considered a black box when somebody sends an email to the WG10 exploder.

Patrice Poyet (CSTB, France) writes in an email of Feb 16 1996:

> My understanding of the situation, and given the fact that a number of AP are facing that reluctance with respect to the benefits supposedly brought by the integration stages, is that we would need a more stepwise process in STEP. It is, if we wish to keep the industry in, of the upmost importance that we offer at least a two stage process, one finishing at the end of group1 with potential ARM based implementations (and an agreed standard at that stage, even though restrictions might apply to it like it does not cover the ultimate goals stated by STEP of interoperability between disciplines), and a second stage, that the industry would be more eager to pay for, if they could already make money with stage1 implementation, which would lead to completely integrated AP, AIM based implementations.

I agree with Patrice, and as a first proposal I would like WG10 to progress with adopting at least the first three of the bullet points given above as a requirement for future APs. This would allow to make an AP a standard which just contains the ARM, as being the first step in an AP development process, and this is as well a necessary precondition for Yuhwei's scenario.

An alternative scenario for AP documentation and implementation

Just to understand the problem better, let's consider another way to define the AP as a standard and another way to implement it, where the application programmer's perspective remains completely unchanged: Consider the AP was just the ARM really (of course we would then no longer call it an ARM, we would find another term), the AP document just contains one model (which is the one we call the ARM now), and the application programmer is using this model as the basis for sending and receiving data, and to implement his application upon it.

Comparing this alternative scenario with the one described above yields some observations. Describing differences, only the case of "using one AP at a time" is considered:

- Within an application based on the AP, a mapping between the neutral data model and the internal data model of the application is needed anyway, because it is a myth to believe that at one point in time all applications of a given functionality are using the same internal data model, since different products have to be optimized for different usage cases, since different versions of the neutral data model will coexist, etc.

- Therefore, the work defining the mapping language has to be done anyway, because at least the mapping between the neutral data model given in the standard and the internal data model of the application has to be specified at some point in time. For this purpose a mapping language on the instance level, suitably linked to EXPRESS and powerful enough, is needed as an international standard.
- Given both scenarios really work — like software works today — from the application programmer's and the user's perspective there is no difference, except that for the currently used scenario the probability of erroneous cases is much higher, because an additional mapping is involved.
- The alternative scenario avoids standardizing any mapping between different data models. The definition of a mapping between data models always involves algorithms whose correctness cannot be verified formally. This means that they have to be verified by being tested against a limited set of test cases. Defining a mapping algorithm within a standard means that time for correcting errors found later is of the order of 5 years.
- The proof is hardly possible that the inverse mapping from AIM to ARM really matches the mapping from ARM to AIM. Having both mappings in a standard, again, the time for correcting errors found later is of the order of 5 years.
- The costs of creating these two mappings (ARM to AIM and AIM to ARM) are representing a considerable amount of money. This amount has to be spent per AP project. Of course, this money is paid by somebody (a company, an industrial consortium, or a public spender) and received at the same time by somebody (typically a highly skilled third sector company or a university), where the interest of the money being spent depends on which side one belongs. But overall, the alternative scenario avoids spending this money, so the scenario we are currently following needs good reasons for being chosen.

Without considering more than one AP at a time, the only conclusion to be drawn can be that the alternative scenario is preferable. There is no reason for introducing the AIM model as a second step, at least for the case of "using one AP at a time".

If WG10 and SC4 would adopt this position in the future, it does not mean that no mapping exercise is allowed anymore. Of course, mapping of a given model onto another one improves the given model. But it is sufficient to standardize the improved model, and there is no reason to standardize the target model and one or two mappings as well, provided that there are no other benefits linked to target model and mappings being standards.

Considering "AP Interoperability"

In Sydney in March 1995, Yuhwei Yang, the ISO/TC184/SC4/WG4 convener, presented her position on "AP Interoperability", which was perceived as the official WG4 position. She admitted that the usage of more than one AP at a time was not intended originally when the concept of Application Protocols was adopted, which I believe is true. Therefore, she admitted, what is called "AP Interoperability" is a scenario not really supported by the current STEP methodology.

In fact, nobody has yet shown a concrete case where it is feasible to create a common database which can be written using one AP and be read using another (the case of “Intersection Interoperability”), or which can be written and read using the same set of APs in their combination (the case of “Union Interoperability”). Not only some conflicting global EXPRESS rules in the APs and other conflicts on the EXPRESS level prohibit such usage, but there is another obstacle which is the fact that there is **no common Conceptual Schema** behind the union of all APs having been developed so far.

The ANSI-SPARC Three Schema Architecture defines clearly the roles of a Conceptual Schema and its corresponding External Schemas (and the role of the Internal Schema which is not important here). Fundamental to this concept is that the semantics of the data types and relations defined is linked to the Conceptual Schema, from where any corresponding External Schema inherits its semantics. Given this semantics is already defined, it may happen that its presentation is refined or restricted within an External Schema, but ANSI-SPARC does not work if the semantics is defined solely on the External Schema level.

Let’s consider an example: In the Conceptual Schema there is the concept of a person with an identification, a name, a birth date and a salary, and the concept of a location with street and postal code etc. The concepts are put together by the following relation:

- a relation between person and location, where instances of the relation mean that the person instance normally works at the location instance as being the person’s office.

Let’s consider an External Schema which projects the whole database to person with name, to location and to the relation between person and location meaning office location, adding a particular presentation to the location in terms of an address template. It is assumed that this External Schema was defined to restrict the database access, but also to allow the distribution of paper mail to all relevant persons.

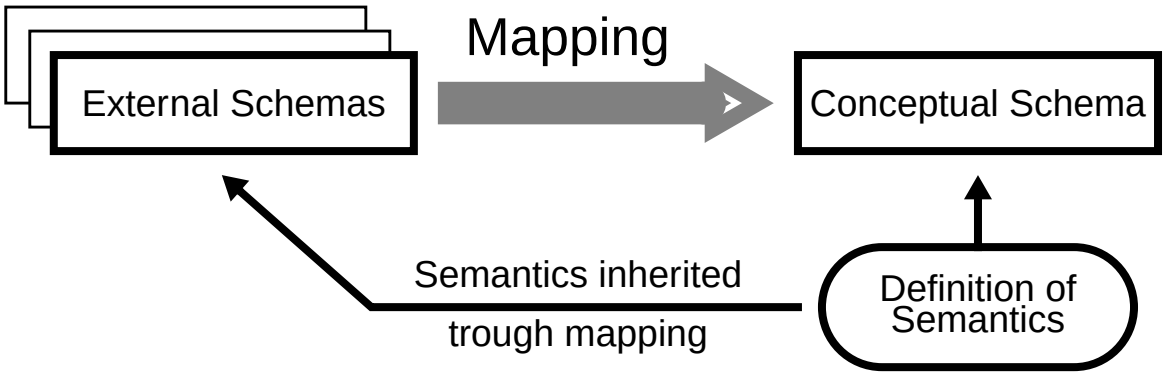
Let’s consider another External Schema which projects the whole database to person with identification, location, and the relation between person and location meaning office location. It is assumed that this External Schema was defined to allow room allocation for the companies employees.

From the viewpoint of both External Schemas, an instance of the person - location relation has the same semantics as it has according to the Conceptual Schema. Without extending the conceptual schema, it is impossible to create another External Schema covering another relation between person and location, which means that the person’s home is at the location.

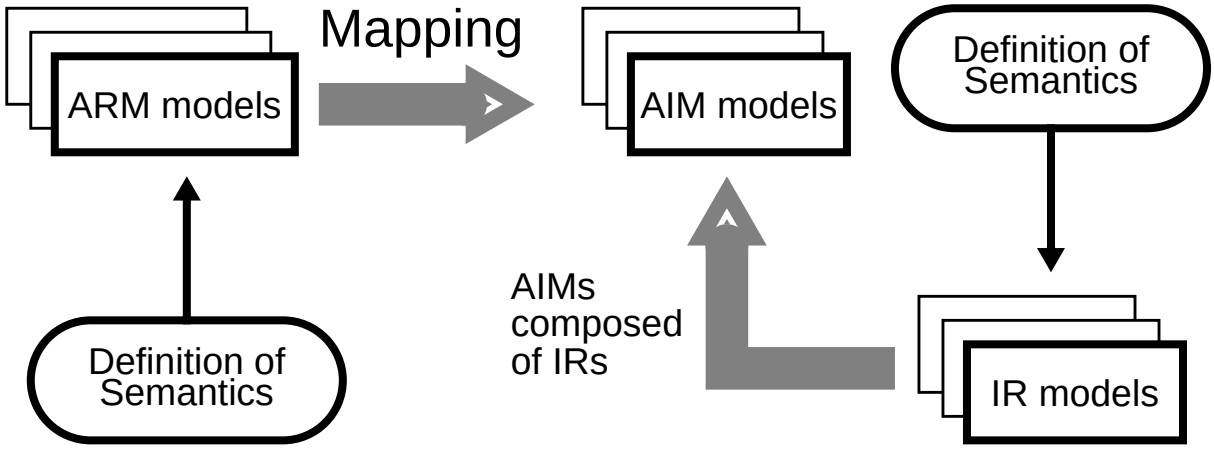
If the Conceptual Schema would be abused to this effect by extending the semantics for the first External Schema, say to allow the distribution of paper mail to somebody at his home, the usage of the second External Schema would be disrupted, because this schema considers any instance of the person - location relation as meaning that the location was the person’s office. Therefore, reassigning rooms to persons, it would erroneously consider the relation instance as an existing room assignment, but it isn’t.

Changing the Conceptual Schema by making it more liberal does also not allow to add new semantics later on. If the Conceptual Schema would just contain a person - location relation template without particular semantics but with an extra string called “description”, adding relation instances meaning home address by the first External Schema would still disrupt the view created by the second External Schema, unless there is a mechanism to distinguish semantics. Such a distinguishing mechanism has to be defined once for all External Schemas, to make sure that no External Schema conflicts with another, hence, the addition of any distinguishing mechanism has to be done on the Conceptual level. It is nothing but a semantic extension of basic concepts.

Summarizing ANSI-SPARC, we can conclude that the definition of semantics has to be done on the Conceptual level. Given semantics defined, it is (to some extent) inherited to the External level where access restrictions (called projections) may apply, but no semantic extension is possible anymore.



In STEP APs we are using a slightly different approach: First a set of constructs is given (Integrated Resources) with partially defined semantics. For each AP separately, they are put together and extended creating the AIM. Finally a mapping to the AIM is defined.



It is crucial that mapping data model A onto data model B implies model B to inherit semantics from model A. It is not the other way round. If for example the concept of a “Cartesian point” is defined as part of model B, we can introduce the concept of “center of a circle” by mapping it to “Cartesian point”. By this mapping, the concept “center of a circle” is defined as a “Cartesian point” which plays a particular role (the concept of playing a role has to be defined in model B already).

This correlation of mapping direction and semantic inheritance direction is the reason why it is not possible to induce the semantics of the AIM model of an AP by defining the semantics of the ARM model and defining an ARM to AIM mapping. Such a mapping would allow to derive ARM semantics from AIM semantics, but not the other way round.

From all these considerations based on ANSI-SPARC we can draw the conclusion that it is impossible to integrate several External Schemas (called APs) into on coherent data model, unless the full semantics of the union of all APs to be integrated would have been already defined within some underlying Conceptual Schema onto which all mappings are defined. Such an underlying **Conceptual Schema does not exis**t, and it cannot exist because its existence would preclude any semantic extension of the STEP standard as a whole.

Therefore, I believe, the usage of more than one AP at a time will never be supported to a useful extent by defining ARM to AIM mappings. If we want to use more than one AP at a time, we have to find another way.

Side Remark: I believe we should examine a scenario, where **semantic building blocks** are defined in terms of EXPRESS schemas, such that the semantics of each schema is fully defined together with the schema, as far as it goes given the context of the schema. For example we can define schemas containing advanced B-Rep geometry given without any semantics going beyond the geometry as such. Another example would be a schema containing all configuration control aspects of any object, not considering the nature of the object itself. Such a schema would contain entities like approval_assignment, approval_person_organization, etc. Constructing Application protocols, such semantic building blocks can be used like our simple EXPRESS data types are used, avoiding the repetition of work done. It the semantics of those building blocks are defined completely — this means that the constructs given therein do not have to be further specialized because they have just structural template characteristics — using them induces a modular structure which can be reflected in implementations. This, I believe, is a means to achieve “AP Interoperability”. The difference between these semantic building blocks and AICs is that an AIC assumes all aspects of the semantics of the real world object represented to be defined, while a semantic building block only considers specific semantics to be defined. The semantic building blocks work like the EXPRESS data type REAL, implying the definition of some specific semantic aspects of the bit array used for its representation to be defined (e. g. REAL addition and REAL multiplication) while other semantic aspects are unspecified (e. g. the physical unit of the value), which are specified by the role the data type REAL is playing.

Disadvantage of our current approach

The first usage of an AP is always its implementation in the context of “using one AP at a time”. The usefulness of AP standards for is scenario is crucial for the success of STEP, while the “AP interoperability” scenario is a further requirement which is also necessary. Considering the “using one AP at a time” scenario, the following disadvantages of our current approach are known:

- Due to the restriction imposed by Interpretation to disallow adding new constructs when defining the AIM schema (with only few exceptions) — together with the other aspects of our approach — some crucial entities in the AIM schema are modeled in terms of a huge set of difficult local rules. These rules mainly restrict the content of attributes or attributes of attributes with respect to their entity data type or their STRING values. This modeling style prohibits compile time checking of correctness of the EXPRESS specification itself, it prohibits to compile the definition of the entity into structural parts of the code generated, and it requires at the same time to run time consuming checking routines at runtime. In addition, it makes the EXPRESS text hard to be read by humans which, together with the compile time checking prohibition, creates a major problem. The problem is in the probability of errors being still contained in the specification when standardized, which have not yet been detected. Compile time checking is not applicable always, but as far as it is, it makes the specification of EXPRESS code much safer.
- Essential parts of the semantics of an EXPRESS construct are always given in terms of natural language, figures or other formalisms like math. Especially the mapping of the data instances to the real world information has to be given this way. Our current approach spreads the semantic description for a given AIM construct over the whole standard: some parts are given together with the short form AIM (clause 5), some part is given in the corresponding IR Part (like Part 41), some part is given together with the ARM where the mapping table is used to relate it to the AIM construct. This spreading makes the capturing of the semantics hard for the implementor (especially implementing a post-processor), it creates a high probability of conflicting descriptions or of missing semantic description with respect to the context the implementor is working with. It happens for example that a construct given in an AIM has an attribute which is neither specified as the image of any mapping, and whose meaning is not defined as part of the short form AIM. For this case the only source of information about its meaning is the corresponding IR Part, where the description cannot be appropriate for many cases, because the context of the IR Part is too general. In the end the implementor lacks information about the meaning of the AIM construct, which leads to misunderstandings during data exchange.

- Consistency rules for an AIM construct are given at several places in several forms: Some are given as EXPRESS constraints in the corresponding IR Part, some are given as informal propositions in the corresponding IR Part, some are given as EXPRESS constraints in an AIC, some are given as EXPRESS constraints in the short form AIM, some are given as informal propositions in the short form AIM, some are given explicitly or implicitly as “mapping rules” in the mapping table. Some EXPRESS constraints are given as local rules, some are given as global rules even though they are not really global. The implementor has to be aware of all these constraints (where many of them would be unnecessary if the EXPRESS constructs were defined directly without being forced to inherit something from the IR Parts), he has to collect them making sure that they do not conflict (which cannot be done by a tool) and he has to implement them. Beside the major burden in terms of (partially unnecessary) implementation costs, this approach is highly error prone, because the reviewer of the standard may fail in finding all of the constraints, missing to detect an incomplete specification or a conflict.
- A consistent set of semantic definitions applied to EXPRESS constructs is never created, as it would form an underlying Conceptual Schema. Very often the information given in a IR Part can only be a structural template without conveying any real world semantics (example: Part 41, clause 2.5.3.3 “A `shape_representation_relationship` is an association between two `representations` where at least one of the `representations` is a `shape_representation`.” In AP 203 this semantics is never appended with some real definition.) What results in terms of an AIM is a mixture of EXPRESS constructs whose semantics is more or less well specified.
- Global Rules have to be used in the AIM to impose constraints enforcing the subtypes defined in the AIM to be the only instanciable entity data types, while the supertype which is imported from an IR Part is made uninstanciable. These rules disrupt the set of data types defined, making it hard to get an overview of all instanciable entity data types. Beside this problem, such global rules prohibit “AP Interoperability” on the EXPRESS level.
- Populating the AIM has to be done in terms of huge instance clusters over which the information known for a particular ARM object is spread. Recollecting this information is a hard job (especially implementing a post-processor), it depends on a set of implicit assumptions and is therefore error prone. I do not expect two implementing teams working independently of each other to be able to write a pre- and a post-processor which supports the exchange of data to a large extent, except for the case of geometry where all semantic information is given at one place (Part 42), and where the definition of the semantics of the constructs is common sense from math, anyway.
- The whole approach is unnecessarily complex and complicated. Life would be much easier if the standardized model of one APs was defined without referring to other models, expect when using another model as an attribute without changing its meaning.

- The maintenance of existing standard documents is a hard job, because there is too much interrelationship and repetitiveness. There is no mechanism to encapsulate the details of a given model from its outside usage, therefore a change of one line in one Part potentially affects all other Parts. Especially those pieces of text describing semantics in terms of natural language, have to be checked manually whenever a change to an existing Part of the standard is put in place. It may turn out that we are not able to maintain our products.

Conclusion

The major disadvantage of the Mapping table approach is that is makes a robust implementation of any pair of pre- and post-processors almost impossible, even for the case of “using one AP at a time”. This is the result of this approach to define the semantics of the constructs used for the data exchange or representation on the wrong level. I the concept of mapping is used, any semantics to be defined shall be given on the conceptual level, this is the level which plays the role of the mapping target. But our current approach defines the semantics on the ARM level, which is the origin of the mapping.

Therefore, I propose the following short term actions. WG10 should resolve that

- ARM models have to be given as text in the AP document !
- ARM models have to be written in EXPRESS !
- ARM models have to be specified completely !
- ARM models have to be implementable !

I propose the following action with a medium term effect to the standardization process. WG10 should

- discuss the issues presented and come to a consensus in terms of a resolution adopted, whether some or all of the issues are considerable problems or not.
- Given the kernel of this paper be considered describing a real problem, WG10 should establish a project to describe the existing problem(s) more precisely.
- Based on a precise description, WG10 should consider alternatives, develop a solution together with a migration plan, and finally help SC4 to implement it.